Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tomislav Slijepčević

Primerjava orodij za upravljanje konfiguracij

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentorica: doc. dr. Mojca Ciglarič

Ljubljana, 2014

To delo je ponujeno pod licenco *Creative Attribution 4.0 International (CC BY 4.0)* (ali novejšo različico). To pomeni, da se tako besedilo, slike, grafi in druge sestavine dela kot tudi rezultati diplomskega dela lahko prosto distribuirajo, reproducirajo, uporabljajo, priobčujejo javnosti in predelujejo, pod pogojem, da se jasno in vidno navede avtorja in naslov tega dela in da se v primeru spremembe, preoblikovanja ali uporabe tega dela v svojem delu, lahko distribuira predelava le pod licenco, ki je enaka tej. Podrobnosti licence so dostopne na spletni strani `http://creativecommons.org/licenses/by/4.0/`.

Izvorna koda diplomskega dela, njeni rezultati in v ta namen razvita programska oprema je ponujena pod licenco *MIT License*. To pomeni, da se lahko prosto distribuira in/ali predeluje pod njenimi pogoji. Podrobnosti licence so dostopne na spletni strani `http://opensource.org/licenses/MIT`.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Področje orodij za upravljanje konfiguracij se je razbohotilo hkrati z razcvetom kompleksnih porazdeljenih infrastruktur, kot so računalniški oblaki. Preglejte področje teh orodij in izberite nekaj najbolj obetavnih. Primerjajte njihove lastnosti in zmogljivosti, nato pa jih praktično preizkusite – uporabite za neko preprosto postavitev. Glede na ugotovitve in vtise iz te postavitve izberite dve orodji, ki ju podrobno preučite in uporabite za kompleksnejši primer, na primer za eno od standardnih postavitev računalniškega oblaka OpenStack. Orodji primerjajte glede na enostavnost namestitve in uporabe, opisni jezik in spekter funkcionalnosti, katerega omogočata.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Tomislav Slijepčević, z vpisno številko **63110317**, sem avtor diplomskega dela z naslovom:

Primerjava orodij za upravljanje konfiguracij

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Mojce Ciglarič,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu prek univerzitetnega spletnega arhiva.

V Ljubljani, 15. septembra 2014 Podpis avtorja:

Zahvaljujem se svoji družini, saj me je spodbujala pri učenju in mi vseskozi stala ob strani. Hvala vam.

Hvala tudi sošolcem, s katerimi sem sodeloval pri skupnih projektih FindToFun, YASA in StandingTable. Bili smo zelo uspešni in zato smo skoraj dobili investicijo. Veselim se novega druženja in poslovnih priložnostih.

Zahvaljujem se tudi mentorici dr. Mojci Ciglarič in dr. Matjažu Pančurju, za zanimivo temo diplomske naloge in pomoč. Pri izdelavi sem se veliko naučil. Hvala tudi za brezplačno kotizacijo za prvo konferenco DevOps v Sloveniji, na kateri sem podrobneje spoznal področje, ki ga zajema moje delo.

Kazalo

Povzetek

Orodja za upravljanje konfiguracij so ovrednotena na dveh praktičnih primerih. S prvim smo hoteli ugotoviti, kolikšen je njihov vstopni prag ali koliko časa je potrebnega za nameščanje orodja in postavljanje nečesa trivialnega, kot je spletni strežnik. Po koncu prvega primera smo izbrali dve najboljši orodji, ki sta se še pomerili v drugem primeru. Naloga zadnjega je bila postavitev večjega sistema, izbrali pa smo spletni oblak OpenStack. Med orodji smo primerjali njune konstrukte in pristope, ki smo jih uporabili. Na koncu smo izbrali zmagovalno orodje, za katero menimo, da je lažje in obeta več.

Ključne besede: DevOps, upravljanje konfiguracij, avtomatizirano nameščanje.

Abstract

Configuration management tools are evaluated on two practical examples. With first we wanted to find out their entry barrier or in other words, we wanted to find out how much time it's needed to install the tool and to build something trivial such as a web server. After that we chose the best two, which would compete on second example. Latter had task to build bigger system and we chose cloud computing software platform OpenStack. We compared tool's used constructs and approaches. At the end we chose the winning tool, based on ease of configuration and important technical properties.

Keywords: DevOps, configuration managment, automated installation.

Poglavje 1

Uvod

Veliko se govori o kulturi *DevOps*. Ime je sestavljeno iz besed *developers* in *operations*, ki označujeta računalniška področja v podjetjih. Njeni tipični predstavniki so sistemski administratorji, razvijalci programske opreme in tisti, ki so odgovorni za njeno kakovost. Kultura zahteva tesnejše združevanje med njimi. Stremi h komunikaciji, sodelovanju in integraciji neodvisnih oddelkov, saj se drugače pojavijo težave. Ponavadi si med seboj prelagajo odgovornost za nastale napake in zato se posledično podaljšujejo razvojni cikli, česar si nihče ne želi.

Kulturno gibanje pripisuje zasluge za nastanek in uspeh številnim orodjem, ki so poenostavila veliko procesov in vpeljala številne nove prakse. Med najpomembnejšimi orodji so zagotovo tista za upravljanje konfiguracij (*Configuration Managment tools* – CM). Njihov cilj je obravnavati računalniško infrastrukturo kot kakršnokoli drugo programsko kodo (*Infrastructure as Code*). To dosežejo z opisom infrastrukture v enega od računalniških jezikov, kot so označevalni, domensko-specifični (*Domain Specific Language* – DSL) ali celo programski jeziki. Ko je infrastruktura opisana, jo lahko preprosto in mnogokrat repliciramo, rezultat bo vedno identičen. S tem se izognemo ročnemu konfiguriranju in vzdrževanju posamičnega računalniškega resursa.

Zdaj se aktivno razvija precej orodij CM, ki se med seboj razlikujejo v paradigmi pisanja, opisnem jeziku, podpori operacijskih sistemov, nameščanju in drugem. Težko je blesteti v vseh pogledih, mi pa želimo izbrati orodje, ki bi upoštevalo

naslednje pogoje: biti mora primerno za začetnika, se aktivno razvija, ima podporo večjih podjetij, opisni jezik je preprost in mnogim že znan ter zadnji zelo pomembni pogoj je težavnost namestitve.

Na tem področju je veliko tekmecev, zato je težko napovedati zmagovalca, podjetja in posamezniki pa nočejo čakati, temveč hočejo čim prej umestiti uporabo orodja med svoje temeljne procese. Želimo jim olajšati izbiro in zato smo štiri vodilna orodja preizkusili na praktičnih primerih. Z vsemi orodji smo postavili spletni strežnik. Po prvem vtisu smo dve izločili, s preostalima dvema pa smo še postavili spletni oblak. Primerjali smo implementacije najpogostejših operacij, s katerimi smo se srečali, in analizirali delovanje orodij. Na koncu smo razglasili zmagovalca.

Poglavje 2

Uporabljena orodja

V nadaljevanju je podrobneje razložen namen orodij za upravljanje konfiguracij in po katerem merilu smo jih izbrali. Nato smo jih ovrednotili po sedanji taksonomiji [3], vendar v okrnjeni obliki. Začetki taksonomije segajo v leta, ko je bil položaj drugačen, od takrat pa so se pojavili obetavni tekmeci, ki bi znali resneje ogroziti starejše predstavnike.

2.1 Orodja za upravljanje konfiguracij

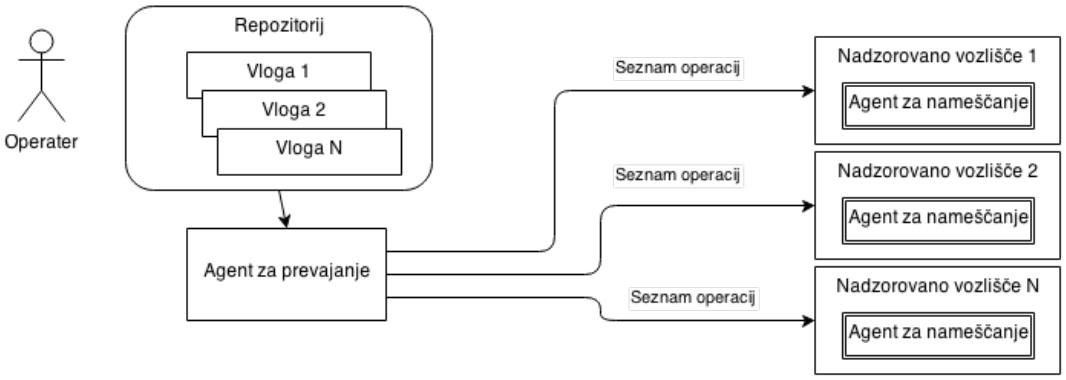
Razloge za njihov nastanek lahko ponazorimo z naslednjim primerom.

Primer Operater upravlja infrastrukturo, ki jo sestavljajo trije računalniki. Naročeno mu je bilo, naj na vse namesti isti spletni strežnik in s tem zagotovi visoko prepustnost njihove spletne strani, saj gre za eno od bolj obiskanih.

Ročna namestitev Odloči se, da jih bo ročno namestil, a se pri enem zalomi, ker je napačno nastavil vrednost ključnega parametra in posledično se strežnik ni odzival. Zdaj mora ugotoviti, v čem se razlikuje od preostalih dveh delujočih. Čez čas dobi stran dodaten obisk in zato je treba dodati še nekaj računalnikov ter jih skonfigurirati, vendar operater pozabi postopek, po katerem se je ravnal pri prejšnjih.

Avtomatizirana rešitev Orodja CM so nastala, da bi odpravila ročno poseganje operaterjev v računalnike, kar povzroča napake. Olajšajo nam tudi repliciranje želenega stanja na več računalnikov. Če bi se operater odločil za avtomatizirano namestitev in bi uporabil orodje CM, bi potekalo nekako takole: v repozitoriju bi imel veliko specificiranih navodil oz. direktiv za razne vloge. Vloga za spletni strežnik bi vsebovala direktive, ki bi poskrbele za namestitev potrebnih paketov, konfiguriranje in zagon servisov ter odpiranje vrat 80 in 443 v požarnem zidu. Operater mora samo povezati želene računalnike z vlogo spletnega strežnika, orodje pa bo apliciralo specificirane direktive, ki so del vloge.

Izbrani računalniki so z vlogami posredovani agentu za prevajanje (*translation agent*), ki vloge prevede v seznam operacij za posamezni računalnik in jih preda agentom za nameščanje (*deploy agents*). Ti operacije sprejmejo in jih izvršijo na ciljnih računalnikih. Proces je viden na sliki 2.1.



Slika 2.1: Klasičen proces orodij CM [3]

Odločil sem se za naslednja orodja CM:

- Puppet (Puppet Labs) [6]
- Chef (Chef Software) [7]
- Salt (SaltStack) [8]
- Ansible (Ansible) [9]

Izbrana so bila, ker gre za najbolj pogosta in najbolj uporabljana orodja za upravljanje konfiguracij [10, 11, 12, 13]. Puppet je najstarejši in najbolj razširjen. Chef je zelo zanimiv, saj uporablja programski jezik Ruby za opisni jezik in je zato samoumevna izbira na tej platformi. Salt se prizadeva za izjemno skalabilnost in hitrost. Ansible je novinec s preprostim opisnim jezikom in je vsaj na videz najpreprostejši med njimi.

Avtomatizirana rešitev - bash skripta Nameščanje se lahko avtomatizira tudi s skripto bash, ki ima napisano pravilno zaporedje ukazov, vendar ima takšna rešitev svoje pomanjkljivosti. Večina namestitvenih skript služi samo nameščanju in so neuporabne za vzdrževanje sistema. Varno se lahko zaženejo le enkrat, nadaljnja izvajanja pa ne zagotavljajo pravilnega delovanja. S ponovnim izvajanjem bi želeli, da se izbriše neustrezne spremembe in uveljavi začetno stanje, vendar žal tega skripte ne zagotavljajo, kot tudi ne zagotavljajo večkratno idempotentno izvajanje.

2.1.1 Taksonomija

Najprej je predstavljena izbrana taksonomija za orodja, ki se precej razlikuje od originalne [3]. Naša se ne spušča v podrobnosti in le izpostavlja najpomembnejše stvari. Lahko bi rekli, da gre za hitri pregled razmer na področju orodij CM. Sestavljena iz naslednjih kriterijev.

Stopnja abstrakcije Z opisnim jezikom lahko dosežemo različne stopnje abstrakcije. Najvišja stopnja sprejme končne zahteve; zagotovi na primer potrebno število poštnih strežnikov, da bo odzivni čas X sekund. Pri najnižji gre le za kopiranje konfiguracijskih datotek za določeno kombinacijo programske opreme in operacijskega sistema. Zapis konfiguracije v takšnih datotekah je pretežno v INI-formatu (de facto standard), veliko pa se uporabljajo tudi razni označevalni jeziki, kot so XML, JSON in YAML.

Vmesnik Orodja ločimo glede na vmesnik za specificiranje konfiguracij. Manj pogosto se uporabljajo grafični vmesniki, kar precej pa je orodij, ki uporabljajo opisne računalniške jezike.

Vrstni red izvajanja Delimo jih na postopkovne in nepostopkovne. S postopkovnimi sistemu velevamo, korak za korakom, kaj mora storiti, zato je vrstni red pomemben. V primeru nepostopkovnega izvajanja je vrstni red stanj znotraj datoteke nepomemben. Paziti moramo le na odvisne direktive. Npr. direktiva, ki zahteva zagon spletnega strežnika, je odvisna od direktive, ki namesti potrebno programsko opremo za spletni strežnik, zato mora prva direktiva počakati z zagonom, dokler poteka namestitev druge direktive.

Domena pokritosti Orodje lahko podpira širok nabor operacijskih sistemov ali pa je omejeno le na enega. Razlikujejo se tudi v podprtih funkcionalnostih; lahko je splošno – namensko in omogoča upravljanje celotnega operacijskega sistema, ali pa omejeno npr. na upravljanje požarnega zidu.

Distribucijski model Agente za nameščanje ločimo na potisne (*push*) in poizvedovalne (*pull*). Če so potisni, jih agenti za prevajanje takoj obvestijo o novih poslih. Če so poizvedovalni, potem periodično sprašujejo agente za prevajanje, ali imajo novi posel.

Stranke Priložen je seznam večjih podjetij, ki uporabljajo orodje. Tudi to je pomemben kriterij pri odločanju. Večje stranke namreč namenijo več finančnih sredstev, kar omogoča lastnikom orodij nadaljnje razvijanje.

2.1.2 Puppet

Stopnja abstrakcije Trenutno omogoča srednjo stopnjo abstrakcije. Konfiguriranje je v principu videti tako: poskrbi, da je nameščen spletni strežnik apache – na operacijskem sistemu Fedora je potreben paket httpd, na sistemu Ubuntu pa apache. Torej gre za konfiguriranje, ki je odvisno od ciljnega sistema in implementacije programa.

Vmesnik Za specificiranje konfiguracij so razvili jezik DSL.

Vrstni red izvajanja Nepostopkovno izvajanje stanj. Za razvrščanje uporabljajo svoj sistem, ki ponuja veliko mehanizmov za določanje odvisnih direktiv.

Domena pokritosti Je splošno – namensko orodje. Podpira večino operacijskih sistemov: AIX, BSD, HP-UX, Linux, OS X, Solaris in Windows.

Distribucijski model Agenti za nameščanje privzeto delujejo na poizvedovalni način, na voljo pa je tudi potisni način.

Stranke Google, Twitter, RedHat, Salesforce, Starbucks, at&t.

2.1.3 Chef

Stopnja abstrakcije Srednja stopnja abstrakcije.

Vmesnik Za specificiranje konfiguracij uporabljajo programski jezik Ruby.

Vrstni red izvajanja Postopkovno izvajanje stanj.

Domena pokritosti Je splošno – namensko orodje. Podpira iste operacijske sisteme kot predhodno orodje Puppet.

Distribucijski model Agenti za nameščanje privzeto delujejo na poizvedovalni način, na voljo je tudi potisni način.

Stranke Facebook, Splunk.

2.1.4 Salt

Stopnja abstrakcije Srednja stopnja abstrakcije.

Vmesnik Za specificiranje konfiguracij uporabljajo množico različnih jezikov. Med bolj znanimi sta označevalna jezika JSON in YAML ter programski jezik Python.

Vrstni red izvajanja Privzeto se direktive aplicirajo nepostopkovno. Za razvrščanje uporabljajo svoj sistem, ki ponuja veliko mehanizmov za določanje odvisnih direktiv. V nastavitvah orodje lahko vključimo postopkovni način.

Domena pokritosti Je splošno – namensko orodje. Podpira nekoliko manj operacijskih sistemov: BSD, Linux, OS X, Solaris in Windows.

Distribucijski model Agenti za nameščanje privzeto delujejo na poizvedovalni način, na voljo je tudi potisni način.

Stranke LinkedIn, Orange, Sky, Apple, Comcast, CloudFlare.

2.1.5 Ansible

Stopnja abstrakcije Srednja stopnja abstrakcije.

Vmesnik Za specificiranje konfiguracij uporabljajo označevalni jezik YAML z dodano semantiko. Npr. z besedama *with_items* in *with_subelements* tvorimo zanke.

Vrstni red izvajanja Postopkovno izvajanje stanj.

Domena pokritosti Je splošno – namensko orodje. Podpira iste operacijske sisteme kot Puppet in Chef.

Distribucijski model Agenti delujejo na potisni način.

Stranke Attlasian, Twitter, EA Games, Spotify, Nasa, Morotola.

2.1.6 Celovit prikaz

Za lažjo primerjavo orodij smo zbrali njihove lastnosti v tabelo 2.1. Izpustili smo seznam strank, da bi lahko tabelo prikazali v celoti na eni strani.

Kriteriji	**Puppet**	**Chef**	**Salt**	**Ansible**
Stopnja abstrakcije	Konfiguriranje je odvisno od ciljnega sistema in implementacije programa.	Konfiguriranje je odvisno od ciljnega sistema in implementacije programa.	Konfiguriranje je odvisno od ciljnega sistema in implementacije programa.	Konfiguriranje je odvisno od ciljnega sistema in implementacije programa.
Vmesnik (za specificiranje konfiguracij)	Opisni jezik - jezik DSL.	Opisni jezik - programski jezik Ruby.	Opisni jezik - označevalna jezika JSON in YAML, programski jezik Python.	Opisni jezik - označevalni jezik YAML.
Vrstni red izvajanja	Nepostopkovno izvajanje.	Postopkovno izvajanje.	Postopkovno in nepostopkovno izvajanje. Privzeto je nepostopkovno.	Postopkovno izvajanje.
Domena pokritosti	Splošno – namensko orodje. Operacijski sistemi: AIX, BSD, HP-UX, Linux, OS X, Solaris in Windows	Splošno – namensko orodje. Operacijski sistemi: AIX, BSD, HP-UX, Linux, OS X, Solaris in Windows	Splošno – namensko orodje. Operacijski sistemi: BSD, Linux, OS X, Solaris in Windows.	Splošno – namensko orodje. Operacijski sistemi: AIX, BSD, HP-UX, Linux, OS X, Solaris in Windows
Distribucijski model (način delovanja agentov za nameščanje)	Privzeto poizvedovalni način, na voljo pa je tudi potisni način.	Privzeto poizvedovalni način, na voljo je tudi potisni način.	Privzeto poizvedovalni način, na voljo je tudi potisni način.	Potisni način.

Tabela 2.1: Celovit prikaz taksonomija in orodij

2.2 Pomožna orodja pri izdelavi

Rešitve za oba primera, postavitev spletnega strežnika in spletnega oblaka, smo razvijali na navideznih računalnikih. Ustvarjali smo jih z orodjem Vagrant [15], ki je v osnovi zelo preprosto. K sebi moraš potegniti zagonsko sliko želenega operacijskega sistema, vendar moraš paziti na ustreznost slike, kajti zgrajene so za določene hipervizorje, ki morda niso nameščeni na lokalnem operacijskem sistemu. Ustvariti je še treba datoteko Vagrantfile, v kateri po meri definiraš stanje navideznega računalnika in nato v ukazni vrstici sprožiš proces ustvarjanja.

Vagrant bo zagonsko sliko in navodila iz datoteke predal hipervizorju, nato bo ustvarjen navidezni računalnik. Za hipervizorja smo izbrali VirtualBox [16], ker je največ zagonskih slik zgrajenih zanj.

2.2.1 VirtualBox in Vagrant

Opisano je delovno okolje, v katerem smo razvijali vloge za orodja CM.

Pri postavitvi spletnega strežnika nismo spreminjali navideznega računalnika in tudi nismo uporabljali naprednejših mehanizmov, ki jih ponuja VirtualBox.

Pri postavitvi spletnega oblaka smo morali definirati stanje navideznega računalnika, ki bi posnemalo oblak v praksi. Sproti smo tudi delali posnetke diska računalnika, kar se je izkazalo za zelo dobro potezo. Namreč implementacija tako velikega sistema, kot je spletni oblak, zahteva veliko poskusov, zato se je se je velikokrat potrebno vračati na začetek oz. na zadnjo točko, ko je oblak še deloval.

Vsebino naše datoteke Vagrantfile za spletni oblak prikazuje primer 2.2. Pomembne so vrstice, ki se začnejo s `config.vm`. Z njimi se spreminja navidezni računalnik. Niz *fedora20* predstavlja simbolično ime računalnika; v našem primeru je poimenovan po nameščenem operacijskem sistemu Fedora [4], 20. verzija. Odpreti je bilo treba vrata v požarnem zidu in zagotoviti 2 omrežji – zunanje in notranje. Nato sledi še del kode, ki zadeva izključno hipervizor VirtualBox. Od njega zahteva 2 procesni enoti, 4 GB delovnega spomina in dodatni trdi disk z velikostjo 4 GB. Če bi uporabljali drug hipervizor, npr. VmWare, bi bilo potrebno nastaviti zadnje parametre posebej zanj.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

VAGRANT_ROOT = File.dirname(File.expand_path(__FILE__))
file_to_disk = File.join(VAGRANT_ROOT, 'filename.vdi')

$script = <<SCRIPT
#!/bin
SCRIPT

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "fedora20"
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.network "private_network", ip: "192.168.33.10"
  config.vm.network "public_network"
  config.vm.provider "virtualbox" do |vb|
    vb.cpus = 2
    vb.memory = 4096
    vb.customize ['createhd', '--filename', file_to_disk, '
      ↪ --size', 4 * 1024]
    vb.customize ['storageattach', :id, '--storagectl', 'SATA
      ↪ ', '--port', 1,
      '--device', 0, '--type', 'hdd', '--medium',
        ↪ file_to_disk]
  end

  config.vm.provision "shell", inline: $script
end
```

Slika 2.2: Vagrant: vsebina datoteke Vagrantfile

Poglavje 3

Postavitev spletnega strežnika

Za začetek smo si izbrali postavitev spletnega strežnika, ker gre za trivialno in pogosto opravilo. Pri tem sta nas zanimali predvsem težavnost namestitve orodja in težavnost postavitve strežnika ter koliko časa je potrebnega za vse skupaj.

Proces je bil takšen:

- namestitev orodja CM
  - nameščanje programske opreme na centralnem vozlišču,
  - morebitno nameščanje opreme na podvozliščih,
  - zagotavljanje komunikacije med centralnim vozliščem in podvozlišči;
- postavitev spletnega strežnika
  - nameščanje spletnega strežnika nginx,
  - dodajanje slike,
  - ustvarjanje uporabnika in skupine,
  - spreminjanje pravic od slike,
  - ustvarjanje statične strani, ki prikazuje sliko,
  - zagon strežnika.

3.1 Vstopni prag orodij

Z izrazom centralno vozlišče mislimo na operaterjev računalnik, z izrazom podvozlišče pa vse računalnike, ki jih operater nadzoruje in na katerih bodo aplicirane direktive.

Puppet Programska oprema se nameša na centralnem vozlišču in podvozliščih. Zadnja zaprosijo centralno vozlišče za komunikacijo, centralno vozlišče pa mora posamezno podvozlišče odobriti, preden steče komunikacija. Nastajale so različne težave pri vzpostavitvi komunikacije. Zbiranje informacij o podvozliščih ni mogoče brez dodatne programske opreme *MCollective* [14], ki pa jo je zelo kompleksno namestiti. Na centralnem vozlišču se ukazi lahko izvršujejo v privilegiranem in neprivilegiranem načinu, vrneta pa različne rezultate; npr. pri ukazu za pregled vzpostavljenih povezav med centralnim vozliščem in podvozlišči smo naleteli na težavo, ko smo ukaz izvajali v neprivilegiranem načinu in smo nepričakovano dobili prazen seznam. Nato smo po daljšem času ugotovili, da je treba ukaz izvršiti v drugem načinu in posledično nam je izvršen ukaz prikazal povezavo z našim podvozliščem. Če bi nam Puppet takoj odgovoril z napako, bi bil problem nemudoma rešen.

Glede opisnega DSL-jezika, ki so ga razvili, so mnenja deljena. Nekateri pravijo, da je jezik preprost in pravšnji za to domeno, vendar vseeno terja nekaj časa za učenje. Težav z vrstnim redom izvajanja direktiv nismo imeli. Puppet namreč razvrsti direktive glede na odvisnost med njimi in pri tako majhnem primeru je odvisnosti zelo malo, zato nismo imeli težav. Predvidevamo, da bi se pri večjem primeru hitro znašli v položaju z veliko odvisnostimi med direktivami in bi zato težje razumeli njihov končni vrstni red izvajanja.

Chef Večji del nalog, ki jih pri tradicionalnem modelu opravlja centralno vozlišče, je prenesenih na t. i. delovno postajo. Torej smo imeli poleg nameščanja na centralnem vozlišču in podvozliščih opravka še z nameščanjem delovne postaje. Celoten postopek je težaven. Izvajanje ukaza *chef-server-ctl* reconfigure, kar je eden od korakov pri nameščanju, ni bilo uspešno, ker je imel Chef težave s pridobivanjem imena gostitelja (hostname). Dobili smo poročilo o napaki, vendar si z njim nismo mogli pomagati. Na spletu smo ugotovili, da gre za pogosto težavo, ki ima

rešitev. Po odpravljeni napaki smo naleteli na nove težave, ki začetnika precej zmedejo in odvrnijo od dela z njim. Tudi njihova terminologija ne pripomore k boljšemu razumevanju, saj uvajajo nove termine, ki le zmedejo začetnika. Termin recept (*recipe*) predstavlja datoteko z direktivami, kuharska knjiga (*cookbook*) predstavlja množico receptov oz. direktiv, terminalsko orodje pa se imenuje nož (*knife*). Poimenovanje je zanimivo, vendar nepotrebno.

Salt Orodje se postavi na isti način kot pri orodju Puppet. Težav pri tem ni bilo. Direktive smo napisali na nepostopkovni način, ki upošteva odvisnosti med njimi in na podlagi tega zgradi vrstni red. Napisane so bile v jeziku YAML. Tudi pri tem nismo imeli večjih težav, vendar menimo isto za vrstni red izvajanja, kar smo napisali za Puppet – da nam lahko oteži pisanje direktiv pri večjih primerih. Na srečo se lahko pri Saltu temu izognemo, saj omogoča tudi postopkovni način izvajanja.

Ansible Postavitev orodja poteka samo na centralnem vozlišču, s podvozlišči pa komunicira prek SSH-protokola. Ker uporablja standardni protokol za komunikacijo, nam na podvozliščih ni treba odpirati dodatnih vrat v požarnem zidu, kar pa ne velja za predhodnike, ki imajo svoje lastne protokole in tudi svoja vrata. Direktive smo napisali v jeziku YAML in se izvajajo v istem vrstni red, kot so zapisane, kar nekoliko olajša miselni proces.

Zmagovalca Za orodje Chef se nismo odločili zaradi namestitvenih težav. Najteže je bilo namestiti orodje Puppet, najlažje pa Ansible. Isto je bilo pri nameščanju spletnega strežnika. Za Ansible smo porabili najmanj časa, za Puppet največ. Ansible se je izkazal za najpreprostejše orodje in zato smo ga izbrali za naslednji podvig. Za drugega kandidata smo izbrali Salt, s katerim nismo imeli težav v primerjavi z orodjema Puppet in Chef.

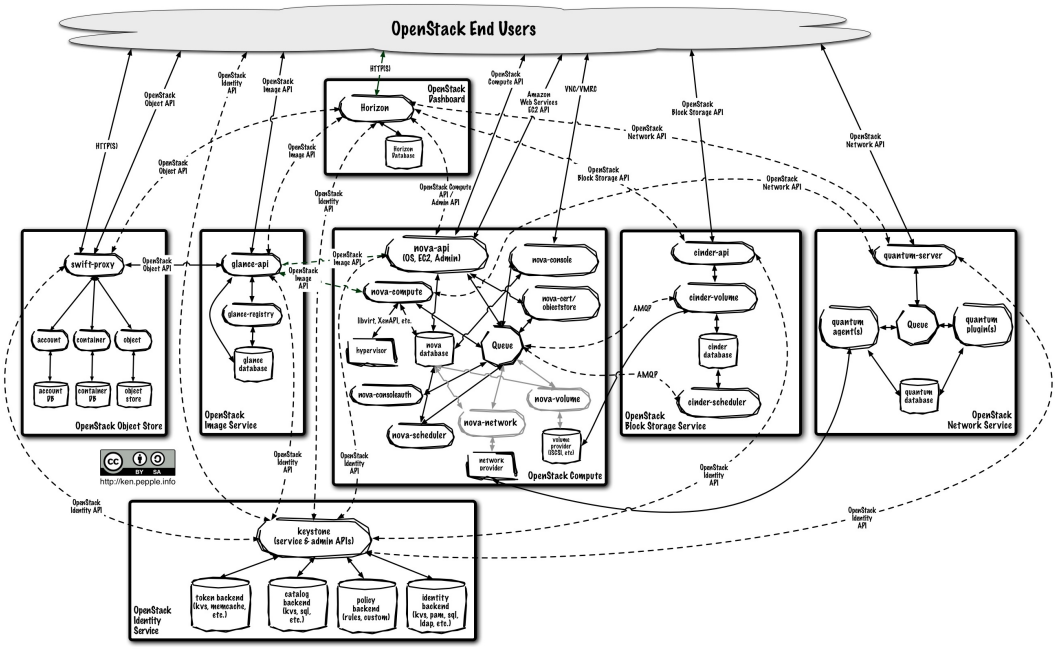
Poglavje 4

Postavitev spletnega oblaka OpenStack

Zanimalo nas je, kako z orodji CM postaviti malo večji sistem, kot je npr. oblak OpenStack [17], ki ima modularno zasnovo. Takšna zasnova nam dopušča, da lahko naloge posameznega modula strnemo v vlogo za orodja CM.

4.1 Zgradba oblaka

Oblak je sestavljen iz več modulov oz. projektov, ki lahko prebivajo na več računalnikih. Tudi njihovi sestavni deli, t. i. servisi, so lahko razpršeni na več računalnikov. Možnosti za postavitev je veliko, mi pa smo se odločili za najpreprostejšo – postavitev na enem računalniku, ki se ji reče *all-in-one*. V praksi se uporablja za testiranje. Postavitev je še posebej pogosta pri razvijanju oblaka, ali ko podjetja oz. posamezniki želijo preizkusiti novo različico oblaka. Je tudi tipična postavitev za začetnike. Oblak je kot celota zelo kompleksen sistem, kar prikazuje slika 4.1. Module je treba pogledati od blizu in spoznati njihove naloge. Šele potem bo vse padlo na svoje mesto in nas dana slika ne bo več prestrašila.

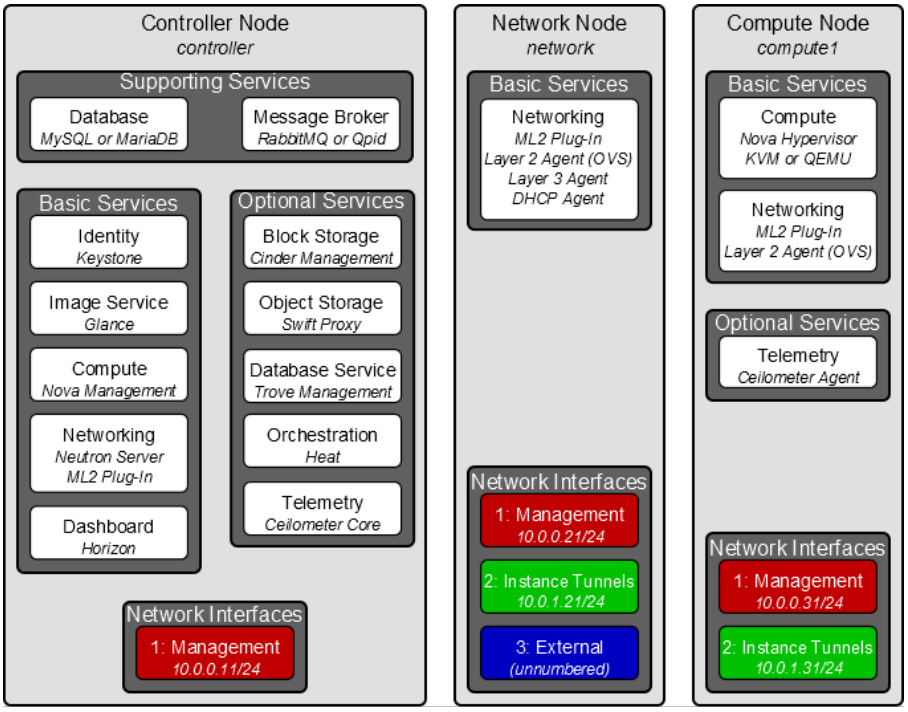


Slika 4.1: OpenStack: spletni oblak [17]

O oblaku se je mogoče pozanimati z branjem uradnih navodil [18] za postavitev na več računalnikih. Napisan je za distribucije RedHat Linux – RHEL, Fedora, CentOS in druge. Distribuciji Fedora in CentOS sta brezplačni in zato zelo ugodni, mi pa smo se odločili za Fedoro, ker navodila od operacijskega sistema CentOS zahtevajo nekaj dodatnih korakov.

Na sliki, na kateri je zasnova oblaka, večje enote predstavljajo module, znotraj modulov pa manjše enote predstavljajo njihove servise. Navodila opisujejo namen posameznega modula in njihovih servisov. Po vsakem opisu so našteti koraki za ročno namestitev: namesti X-paket, spremeni konfiguracijo za servisa Y1 in Y2 od paketa X, omogoči zagon servisa Y1 ob zagonu operacijskega sistema, zaženi oba servisa takoj ...

Končna postavitev je prikazana na sliki 4.2. Na njej so tri enote, ki predstavljajo tri računalnike, znotraj katerih so našteti moduli oz. servisi. Najprej so opisani najpomembnejši moduli in nato so razložene enote na sliki.



Slika 4.2: OpenStack: postavitev iz uradnih navodil [18]

Predpriprave Večina servisov potrebuje podatkovno bazo za shranjevanje informacij. Prav tako potrebujejo sporočilno vrsto (*Message Queue*) za koordiniranje operacij in izmenjevanje statusnih sporočil. Najpogostejša izbira za bazo je MySQL oz. MariaDB. Za sporočilno vrsto se najpogosteje izbira med Qpid in RabbitMQ.

Keystone Odgovoren je za identifikacijo in avtorizacijo (*Identity Service*). Je centralni in nepogrešljiv modul. Vzdržuje seznam uporabnikov in njihove pravice. Navzven izpostavlja katalog dostopnih storitev in njihove programske vmesnike (*API*). V ta katalog mora večina modulov dodati svoje storitve.

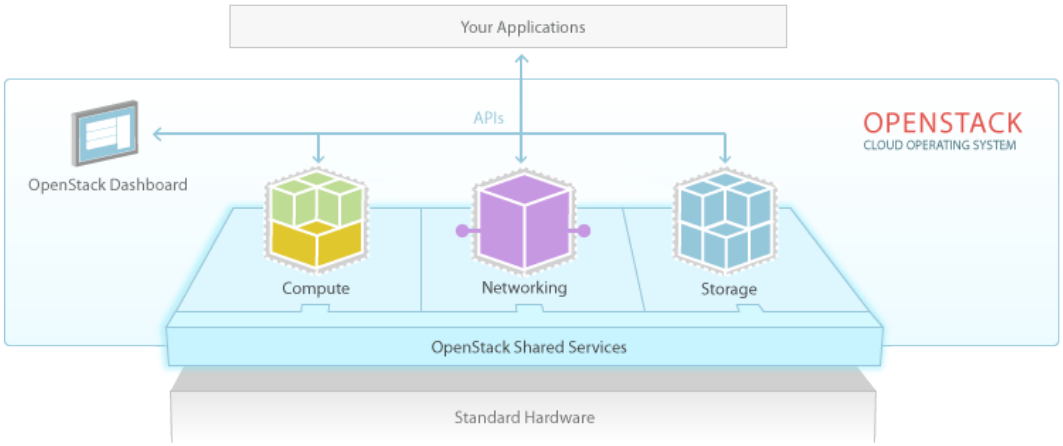
Glance Odgovoren je za zagonske slike (*Image Service*) in zanje ima shrambo, po kateri je mogoče iskati, pridobivati in shranjevati nove slike. Isti nabor operacij ponuja tudi za metapodatke o slikah.

Nova Je gonilo oblaka. Odgovoren je za poganjanje navideznih računalnikov (*Compute Service*). Zasnovan je za horizontalno skalabilnost na standardni strojni opremi.

Neutron Odgovoren je za navidezno mreženje navideznih računalnikov in nadzorovanje fizične mrežne opreme, ki jo izrablja za zunanjo povezljivost (*Networking Service*). Navidezno mreženje omogoča kreiranje abstraktnih elementov, ki izvirajo iz fizičnega sveta. Ti elementi so: omrežja, podomrežja in usmerjevalniki, požarni zidovi, naprave za porazdeljevanje prometa (*Load Balancer* – LB) in VPN-omrežja.

Cinder Odgovoren je za upravljanje nosilcev podatkov (Block Storage). Navideznim računalnikom ponuja nosilce podatkov in ustvarjanje posnetkov nosilcev (*Volume snapshots*).

Horizon Gre za spletni vmesnik oz. spletno stran, ki omogoča administratorjem in uporabnikom dostop do resursov in storitev oblaka. Zaradi tega je interakcija z oblakom zelo poenostavljena. Če tega modula ne namestimo, smo prisiljeni dajati ukaze v terminalu, kar je lahko zamudno za nevešče uporabnike.



Slika 4.3: OpenStack: najpreprostejši prikaz [17]

Na sliki 4.3 so prikazani vsi moduli. Ime *Compute* je drugo ime za modul

Nova, modula Keystone in Glance pa sta označena z izrazom *Shared Services*. Slika prikazuje povezanost modulov, lahko pa tudi predstavlja dejansko postavitev oblaka, pri čemer simboli predstavljajo računalnike ali računalniške gruče. Torej bi imeli računalnike za storitve Compute (Nova), Networking (Neutron) in Storage (Cinder), druge deljene storitve pa bi prebivale na kontrolnem računalniku (*Controller*). Podobno je postavljeno v navodilih, kot kaže slika 4.2 – storitvi Compute in Neutron imata svoj računalnik, storitev Cinder pa je prestavljena na kontrolni računalnik. Potem je najbrž očitno, kaj se zgodbi oz. kje prebivajo storitve, če imamo samo en računalnik - vse se preselijo na kontrolni računalnik. Tako smo si tudi zamislili svoj primer. Naj še omenimo, da ima naš kontrolni računalnik eno mrežno kartico za notranje omrežje in drugo mrežno kartico za zunanje omrežje. Notranje je namenjeno storitvam OpenStack, zunanja mrežna kartica pa nam zagotavlja zunanjo povezljivost, ki je nujna, če želimo od zunaj dostopati do navideznih računalnikov.

Oblak smo postavili z napisanimi direktivami za obe orodji – Ansible in Salt. Srečali smo se z raznimi konstrukti in pristopi, ki se razlikujejo med orodjema in zato smo jim namenili naslednje podpoglavje.

4.2 Sestavni deli orodij

Sledi opis konstruktov in pristopov, ki so bili uporabljeni za postavitev spletnega oblaka. Pri vsakem so priloženi primeri za obe orodji. Najprej je razložena rešitev za Ansible, nato za Salt, za konec pa so na kratko zapisana opažanja za obe rešitvi.

4.2.1 Pisanje vloge

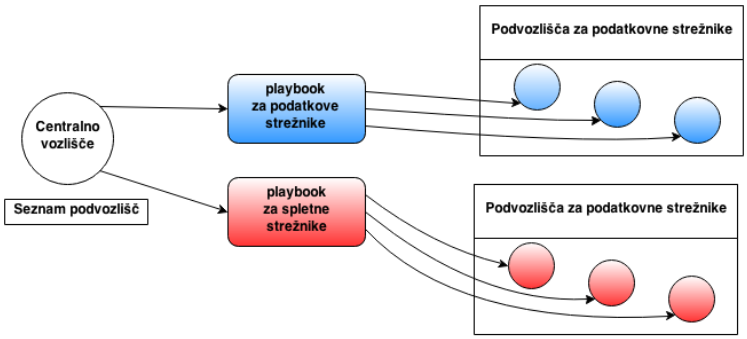
Če so direktive namenjene zgolj enemu primeru uporabe, kot je npr. postavitev spletnega ali podatkovnega strežnika, potem taki celoviti rešitvi rečemo vloga. Morda uporabljeni termin ni ravno najboljši. Ansible ga uporablja, Salt pa temu pravi formula.

Vloge določamo vozliščem, ki jih nadzirajo orodja za upravljanje konfiguracij. Vozlišča predstavljajo računalniške resurse – fizične ali navidezne računalnike. Obe orodji omogočata tudi uveljavitev vlog prek kategorizacije vozlišč, kot je prikazano

s primerom 4.4 za Ansible, vendar isti princip velja tudi za Salt. Na primeru so vozlišča prek naslovov IP kategorizirana kot spletni strežnik (web) ali podatkovni strežnik (database server – db). Po opravljeni kategorizacija se poveže posamezno kategorijo z vlogo oz. direktivami in nato sledi njihova uveljavitev, kot je ponazorjeno na sliki 4.5.

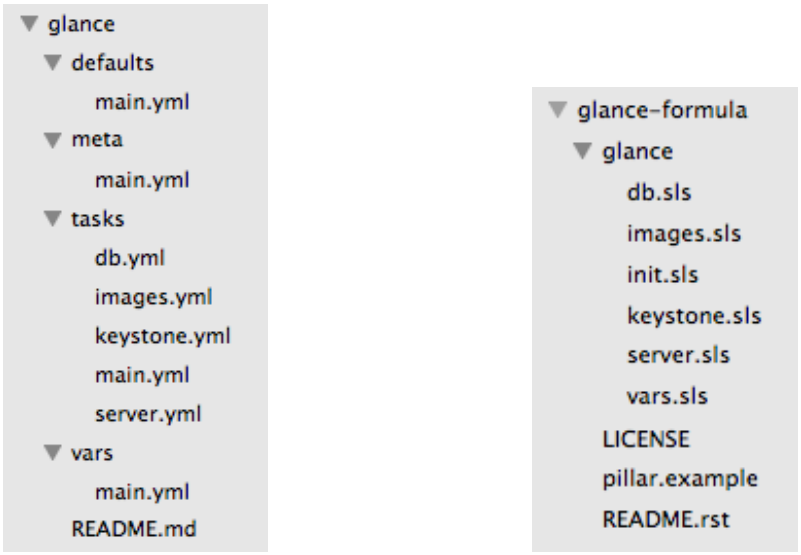
```
[web]
192.168.10.100
192.168.10.101

[db]
192.168.20.100
192.168.20.101
```

Slika 4.4: Ansible: kategorizacija vozlišč



Slika 4.5: Ansible: uveljavitev vlog [5]

Obe orodji imata predvideno zgradbo za vloge. Pokazali bomo vlogo *glance* oblaka OpenStack. Ansiblova zgradba je vidna na sliki 4.6a, Saltova pa na sliki 4.6b.



(a) Ansible: vloga glance (b) Salt: vloga glance

Slika 4.6: Zgradba vloge glance za obe orodji

Spremenljivke Ansible zahteva zapis v datotekah, ki so znotraj direktorija `defaults` in `vars`. Saltu je vseeno, le pravo pot do datoteke moraš navesti pri uporabi; v našem primeru so vse spremenljivke zbrane v datoteki {`IME PROJEKTA`}`/vars.sls` oz. `glance/vars.sls`.

Direktive Obe orodji delujeta na isti princip. Najprej se prebere začetna datoteka z direktivami in nato še vse preostale datoteke, ki so naštete znotraj nje. Ansiblova začetna datoteka je na poti `tasks/main.yml`, Saltova pa na poti {`IME PROJEKTA`}`/init.sls`. Vse vloge, razen vlog Horizon in Keystone, imajo podoben postopek nameščanja in zato tudi podobne direktive, ki smo jih razvrstili v tri datoteke:

- db – začetna inicializacija v podatkovni bazi,
- server – nameščanje modula,
- keystone – integracija z modulom Keystone.

Vsaka vloga ima še dodatne datoteke za svoje potrebe. Tako zgleda seznam v začetnima datotekama za vlogo glance:

- db – poskrbi, da sta na podatkovnem strežniku ustvarjena privilegiran uporabnik in podatkovna baza za modul glance,
- server poskrbi:
  - da so nameščeni potrebni paketi,
  - da so servisi pravilno skonfigurirani,
  - da so servisi zagnani;
- keystone – poskrbi, da je vse ustvarjeno v centralni oblačni enoti Keystone in registrirano v njenem katalogu storitev. Treba je:
  - ustvariti uporabnika z imenom glance,
  - mu nastaviti administratorske pravice,
  - registrirati storitev glance in navesti dostopno programsko točko (API);
- images – dodatna datoteka z direktivami, ki poskrbijo, da shramba glance hrani tudi demonstracijsko sliko z operacijskim sistemom Cirros.

Pogoji in primeri uporabe Za Ansible se večino informacij o vlogi zapiše v datoteko `meta/main.yml` – vrsto licence, podprtost operacijskih sistemov, avtorja in krajši opis. Primer uporabe se napiše v datoteko `README.md`, kar je tudi današnji standard za programsko opremo. Za Salt se vrsta licence zapiše v datoteko `LICENSE`, primer uporabe v datoteko `pillar.example`, v datoteki `README.rst` pa se ponavadi dodatno razložijo stanja. V to datoteko se tudi napišejo informacije o avtorju in podrte operacijske sisteme.

Opažanja Direktive se hranijo in vključujejo na isti način. Poleg zapisa informacij, ki ne zadevajo delovanja vloge, je opaziti še dve veliki razliki – število direktorijev in lokacije spremenljivk. Ansible direktorije uporablja za semantično razvrščanje stvari in uveljavitev konvencij, ki se jih morajo uporabniki držati (Convention over

Configuration), medtem ko Salt ne zahteva prav ničesar. Kateri način je boljši, je stvar posameznika – ali ima raje svobodo, ki jo Salt ponuja, ali uveljavljeno strukturo, ki jo Ansible predpisuje. Razlika pri spremenljivkah je posebej obravnavana v svojem podpoglavju.

4.2.2 Nameščanje paketov in upravljanje servisov

Pogosta operacija, s katero se srečamo na svojih operacijskih sistemih, je nameščanje paketov. Na distribucijah Linux se navadno uporabljajo orodja za upravljanje paketov, ki so napisana za ukazno vrstico, obstajajo pa tudi grafični vmesniki. Orodja vsebujejo seznam repozitorijev, po katerih poizvedujejo in ob zadetku snamejo paket, nakar orodje poskrbi, da je nameščen skupaj z odvisnimi paketi. Bolj znani orodji sta yum za distribucije Linux, zasnovane na operacijskem RHEL sistemu, in apt-get za distribucije, zasnovane na Debianu.

Z upravljanjem računalniških servisov se navadni uporabniki ne srečujejo ravno pogosto, ker so vsi paketi, ki jih nameščajo, navadno takšni, da je za njihove servise samodejno poskrbljeno po nameščanju. Drugače je pri naprednejših uporabnikih, ki se pogosto poslužujejo ukazne vrstice za zagon ali ponovni zagon servisov, kot so httpd za spletni strežnik ali mysqld za podatkovni strežnik.

```
- yum: name=openstack-keystone

- service:
    name='openstack-keystone'
    state=started
    enabled=yes
```

(a) Ansible

```
openstack-keystone:
  pkg:
    - installed
  service.running:
    - enable: True
    - require:
      - pkg: openstack-
        ↪ keystone
```

(b) Salt

Slika 4.7: Paketi in servisi

Primer Pri postavitvi oblaka je treba od modulov najprej namestiti centralni modul Keystone, ki je odgovoren za avtentikacijo in avtorizacijo uporabnikov. Eden od prvih korakov nameščanja je viden na primeru 4.7. Najprej je treba potegniti paket z orodjem za upravljenje paketov, ga namestiti, zagnati istoimenski servis in poskrbeti, da servis teče ob zagonu operacijskega sistema.

```
openstack-keystone:
  pkg:
    - installed

openstack-keystone:
  service.running:
    - enable: True
    - require:
      - pkg: openstack-
        ↪ keystone
```

(a) Salt: konflikt z id-ji

```
install_keystone_package:
  pkg.installed:
    - name: openstack-keystone
start_and_enable_keystone_service:
  service.running:
    - name: openstack-keystone
    - enable: True
    - require:
      - pkg: openstack-keystone
```

(b) Salt: alternativna rešitev

Slika 4.8: Salt: dodatni primeri za pakete in servise

Opažanja Vsaka direktiva ima navedeno ime in tip resurs ter želeno stanje. Resursi so v tem kontekstu stvari, ki jih najdemo na računalniku. To so npr. paketi, servisi, datoteke in drugo. Direktiva za spletni strežnik je takšna:

- ime: httpd,
- tip: servis (service),
- stanje: zagnan (*running* ali *started*).

Ime in tip sta ponavadi napisana čim višje, opis stanja pa pod njima.
Kot je razvidno s primera, znata obe orodji delati z resursi, ki so paketi in servisi. To je tudi samoumevno, saj bi bila orodja CM brez tega precej neuporabna. Pri jeziku YAML, ki je izbran za obe orodji, opazimo različen način naštevanja direktiv. Za Ansible so naštete v seznamu, za Salt pa v razpršeni tabeli. Razlikuje se tudi njihov zapis.

Ansible Ansible (4.7a) ima dve direktivi. Tip resursa prve direktive je označen z besedo `yum` in predstavlja orodje za upravljanje paketov, kar pomeni, da je tip direktive pravzaprav paket. Če direktiva nima navedenega stanja, bo aplicirano privzeto stanje resursa. V primeru paketnega resursa bo direktiva privzeto poskrbela, da je paket nameščen. Drugi resurs je za servisnega tipa in ima navedeno stanje. Od imenovanega servisa zahteva, da je zagnan takoj zdaj in da se zažene ob zagonu operacijskega sistema.

Salt Pri Saltu imajo direktive imena, ki so poljubna in napisana na začetku. Edino, na kar moramo paziti pri izbiranju imen, je njihovo podvajanje, kajti Salt jih razume kot enolični identifikator (*ID*) in zato ne dopušča poimenovanja direktive z imenom, ki je že uporabljeno. Za ime se navadno izbere ime resursa. Če imamo več direktiv, ki opisuje stanje istega resursa, smo naleteli na konflikt z ID-ji (4.8a). Salt ima dve rešitvi za takšne primere. Pri prvi prestavimo ime resursa v opis stanja, kot na primeru 4.8b. Ime je napisano z atributom `name`. Druga rešitev je združevanje direktiv v eno samo, kot smo to storili v našem primeru 4.7b. Čeprav je videti kot ena direktiva, sta v osnovi še vedno dve. Direktiva za servis je odvisna od direktive za paket, saj servis ne more delovati, če paket ni nameščen. Zato moramo v direktivi za servis ta pogoj oz. odvisnost navesti in to se stori z dodatnim atributom `require`, ki zahteva, da se predhodno izvedejo direktive iz seznama. Razlog za navedbo pogoja je v vrstnem redu izvajanja direktiv. Povedali smo že, da Salt omogoča postopkovno in nepostopkovno izvajanje, pri postavitvi oblaka pa smo direktive pisali za nepostopkovno izvajanje, ker je to tudi privzeto. Od nas zahteva dodajanje odvisnosti, da bi pravilno zgradilo končni vrstni red direktiv. Ansible tega ne potrebuje zaradi postopkovnega izvajanja, zahteva pa pravilni vrstni red in zato moramo poskrbeti, da je direktiva za paket pred direktivo za servis.

4.2.3 Pogojni klici

Orodja CM ne znajo delati z vsemi resursi in zato je treba za nepodprte poseči po ukazni lupini. V njej je mnogo vgrajenih ukazov, s katerimi lahko obdelamo resurse. Edini problem je zagotavljanje idempotentnosti ukazov, kar pomeni, da se lahko večkrat izvršijo brez posledic. Večina vgrajenih ukazov ni takšnih in zato je treba

pred njihovo izvršbo nekaj dodatnih korakov. Vzemimo za primer neidempotentni ukaza `mkdir` za kreiranje direktorijev. Orodja CM znajo delati z direktoriji, toda recimo, da to ne drži in smo primorani uporabljati zunanji ukaz mkdir. Na disku želimo imeti na dani poti direktorij. To bi bilo naše želeno stanje. Ukaz nam dovoli kreiranje le, če je dana pot prosta, v nasprotnem primeru pa sproži napako. Torej ponovno izvajanje ukaza ni brez posledic in zato je treba pred ponovnim izvajanjem preveriti, ali je pot prosta.

Pogosto smo imeli opravka z nepodprtimi resursi, zato smo zanje uporabili ukazno lupino. Zaradi njihove neidempotentne narave smo jim morali dodati nekaj predpogojev, da bi dosegli idempotentnost. Predpogoje imata obe orodji, vendar se njuni implementaciji razlikujeta.

Primer Za modul *cinder* je treba registrirati nosilec podatkov, iz katerega bo ustvarjenih več manjših nosilcev za potrebe navideznih računalnikov. Če je glavni nosilec že registriran, ga ne smemo ponovno registrirati.

Ansible Rešitev je vidna na primeru 4.9. Prva direktiva pokliče zunanji ukaz `pvs` za izpis registriranih nosilcev in nato seznam posreduje ukazu `grep`, ki iz njega izlušči nosilec z imenom iz spremenljivke `CINDER_VOLUME`. Ukaz `grep` se zaključi oz. vrne z nekim številom kot vsi zunanji ukazi; število 0 predstavlja uspeh, vsa druga pa neuspeh. Direktive za Ansible omogočajo shranjevanje raznih stvari ob zaključenem izvajanju in med njimi so tudi vrnitvena števila. Shranjene stvari se lahko nato uporabijo v preostalih direktivah, kar smo tudi storili. V drugi direktivi kličemo zunanji ukaz `pvcreate`, ki ni idempotenten. Zaradi tega smo za pogojno izvajanje uporabili vrnitveno število prejšnje direktive. Predpogoj je zapisan z atributom `when` in v njem preverimo, ali se je shranjeno vrnitveno število prejšnje direktive zaključilo z neuspehom, kar bi pomenilo, da nosilec še ni registriran in da lahko nadaljujemo izvajanje glavnega ukaza direktive. Pri vnovičnem apliciranju direktiv bo nosilec že registiran in ga bo zato prva direktiva našla. Posledično bo vrnitveno število vrednosti 0, predpogoj druge direktive pa ne bo izpolnjen in zato se bo direktivo preskočilo. S tem smo zagotovili idempotentno izvajanje.

```
# Check if physical cinder volume created
- shell: pvs | grep {{ CINDER_VOLUME }}
  register: doesCinderPhysicialVolumeExist
  failed_when: False

# Create physical cinder volume
- command: pvcreate {{ CINDER_VOLUME }}
  when: doesCinderPhysicialVolumeExist.rc != 0
```

Slika 4.9: Ansible: pogojni klic

Salt Zunanji ukazi in predpogojna logika so uporabljeni in postavljeni na isti način kot pri Ansiblu. Razlikujeta se le v načinu zapisanega predpogoja. Ansible v predpogojih nudi uporabo shranjenih stvari iz prejšnjih direktiv, Salt pa izvajanje v ukazni lupini. Zaradi slednje Saltove funkcionalnosti smo v predpogoju neposredno uporabili ukaza `pvs` in `grep`. Rešitev je vidna na primeru 4.10.

```
# Create physical cinder volume if doesn't exist
pvcreate {{ CINDER_VOLUME }}:
  cmd.run:
    - unless: >
        pvs | grep {{ CINDER_VOLUME }}
```

Slika 4.10: Salt: pogojni klic

Opažanja Saltov način je nedvomno boljši, saj lahko predpogojno logiko zapišemo neposredno v isto direktivo in se s tem izognemo pisanju dodatne direktive.

4.2.4 Definiranje spremenljivk

Obe orodji imata kompleksen in različen sistem za definiranje spremenljivk, zato smo se omejeli le na spremenljivke, ki se definirajo znotraj vlog.

Primer Večina modulov ima v vlogi spremenljivki `PASS` za dostop do modula Keystone in `DBPASS` za dostop do svoje podatkovne baze. Prikazali smo ju za modul glance.

Ansible Spremenljivke ima shranjene znotraj direktorijev `defaults` in `vars`. Tiste, ki so v direktoriju `defaults`, imajo najnižjo precendenco, zato jih lahko povozijo istoimenske spremenljivke, ki so definirane na katerikoli drug način. Skupaj je načinov več kot deset. Tiste, ki so v direktoriju `vars`, imajo precej višjo precendenco. Velja nenapisano pravilo, da se v direktorij `vars` daje spremenljivke, ki verjetno ne bodo spremenjene, medtem ko se v direktorij `defaults` dajejo spremenljivke s privzetimi vrednostmi, ki bi morale biti zamenjane. Vrednosti za spremenljivki `PASS` in `DBPASS` sta zelo pomembni in zaradi varnostnega vidika ju je treba vsakič zamenjati. Zato je nekako smiselno, da ju vstavimo v direktorij `defaults`. Vsebina datoteke `defaults/main.yml` je vidna na primeru 4.11.

```
GLANCE_DBPASS: glance-db-pw
GLANCE_PASS: glance-pw
```

Slika 4.11: Ansible: spremenljivke v vlogi glance

Salt Ne ponuja semantičnega razvrščanja spremenljivk, kot to počne Ansible, temveč prepušča proste roke pri definiranju spremenljivk. Definirati jih je treba z jezikom Jinja2 [19]. Vsebina datoteke `vars.sls` je vidna na primeru 4.12.

```
{% set GLANCE_DBPASS = 'GLANCE_DBPASS' %}
{% set GLANCE_PASS = 'GLANCE_PASS' %}
```

Slika 4.12: Salt: spremenljivke v vlogi glance

Opažanja Ansible je zahtevnejši glede definiranja spremenljivk, kar nas morda zmoti na začetku, vendar se hitro navadiš in ti je potem samoumevno, kam postaviti vsako novo. Pri Saltu je moteče, da jih ne moramo definirati v jeziku YAML, v katerem pišemo direktive.

4.2.5 Kopiranje predlog

V tem podpoglavju si bomo pogledali, kako se prekopira predloge. Gre za datoteko, ki ima med svojim besedilom tudi imena spremenljivk. Preden se prekopira, gre skozi predprocesor Jinja2 in končni rezultat je datoteka s prvotnim besedilom, pri čemer so imena spremenljivk zamenjana z njihovimi vrednostmi.

Primer V domačem direktoriju si želimo imeti datoteko, ki bi vsebovala podatke za administratorski dostop do modula *Keystone*. Takšna datoteka je lahko le zapisana kot predloga na operaterjevem računalniku, saj so administratorski podatki vsebovani v spremenljivkah in ne v končnih datotekah.

Ansible Za predloge je namenjen direktorij `template` znotraj vloge. Tam smo shranili našo predlogo, njeno kopiranje pa ne bi moglo biti preprosteje (4.13). Direktivi podamo samo ime predloge (`src`) in želeno destinacijo (`dest`).

```
- template:
    src='admin-openrc.sh'
    dest='~/admin-openrc.sh'
```

Slika 4.13: Ansible: kopiranje datoteke

Salt Za predloge nima namenjenega direktorija, navadno pa se vstavijo v direktorij `files` znotraj vloge. V direktivah je treba navesti njihovo absolutno pot. Treba je tudi izbrati in navesti ime predprocesorja, ker jih ima Salt več. Mi smo izbrali Jinja2, ki je bil prav tako uporabljen pri Ansiblu.

```
/root/admin_openrc.sh:
  file.managed:
    - source: salt://keystone/files/admin_openrc.sh
    - template: jinja
```

Slika 4.14: Salt: kopiranje datoteke

Opažanja Tilda operator (~), ki predstavlja domači direktorij trenutnega uporabnika, je podprt samo pri Ansiblu. Za Salt smo morali navesti absolutno pot do domačega direktorija. Tudi za izvorno predlogo smo morali podati absolutno pot, medtem ko ima Ansible konkretno mesto za predloge in je zato dovolj, če mu podamo samo ime predloge, ki je znotraj direktorija.

4.2.6 Spreminjanje INI konfiguracijskih datotek

Servisi imajo velikokrat datoteke, s kateri konfiguriramo njihovo delovanje. To so t. i. konfiguracijske datoteke. Imajo jih tudi servisi spletnega oblaka OpenStack, zapisane pa so v formatu INI. Za lažje konfiguriranje servisov oz. za lažje spreminjanje vrednosti v datotekah INI obstaja orodje crudini. OpenStack ga tudi uporablja, vendar pod drugim imenom – openstack-config. To je bilo uporabljeno v našem primeru. V nadaljevanju je pri uporabi besede datoteka mišljena konfiguracijska datoteka.

Primer in potencialne rešitve Kako spremeniti vrednosti v datotekah? Najpreprostejša rešitev bi bila, da bi enkrat napisali končno stanje datoteke in jo nato le razmnožili. Alternativna rešitev je seznam vrednosti, po katerem se sprehodimo in sproti preverimo prisotnost posamezne. Če se vrednost iz datoteke ne ujema z vrednostjo iz seznama, potem jo prepišemo z vrednostjo iz seznama. Pri preprostejši rešitvi lahko preverimo zgoščeno vrednost datoteke. Če se ne ujema, potem prepišemo datoteko. Ta akcija nam prinese le 1 bit informacije. Z drugo rešitvijo pridobimo informacije o spremembah vseh vrednosti oz. v čem se prejšnje stanje razlikuje od sedanjega. Posledično lahko najdemo vzrok za napačno delovanje servisa. Za isti rezultat bi lahko uporabil znana orodja diff, ki prikažejo razliko med datotekama.

Odločili smo se za 2. rešitev, ki pa je ni bilo težko implementirati pri obeh orodjih.

Primeri implementacije so pokazani iz vloge za modul *Neutron*. Obe implementaciji (4.17, 4.19) uporabljata klicne resurse.

Ansible Spremenljivka `neutron_config` (4.15) vsebuje seznam vrednosti za več datotek, zato je potreben malce kompleksnejši obhod z atributom `with_sub-elements` (4.17). Nato se v ukazni lupini pokliče pomožno orodje (4.16) za vsako vrednost. Orodje primerja staro vrednost z novo. Če se ne ujemata, potem zapiše novo in sporoči, da je prišlo do spremembe. Pomožno orodje je razvito, ker Ansible ne omogoča pogojnih klicev znotraj ukazne lupine. Če bi jih omogočal, bi v pogoju napisali ukaz za preverbo ujemanja vrednosti, glavni ukaz pa bi prepisal staro vrednost, če ne bi bilo ujemanja. Z uporabo pomožnega orodja smo se izognili pisanju dveh direktiv za Ansible.

```
neutron_config:

  - path: /etc/neutron/neutron.conf
    values:
    - 'database␣connection␣mysql://neutron:{{␣NEUTRON_DBPASS
      ↪ ␣}}@controller/neutron'
    - DEFAULT auth_strategy keystone
    ..

  - path: /etc/neutron/plugins/ml2/ml2_conf.ini
    values:
    - ml2 type_drivers gre
    ..
```

Slika 4.15: Ansible: vrednosti za konfiguracijske datoteke

```
#!/bin/sh

VALUE="${@:2}"
ARGS=${@:1:$#-1}

prev_value=$(openstack-config --get $ARGS)

if [[ $prev_value == $VALUE ]]; then
        echo 0
else
        openstack-config --set $ARGS $VALUE
        echo 1
fi
```

Slika 4.16: Pomožno orodje za spreminjanje konfiguracijskih datotek

```
- command: "~/openstack-config-helper.sh {{item.0.path}} {{
    ↪ item.1}}"
  register: res
  changed_when: res.stdout == '1'
  with_subelements:
    - neutron_config
    - values
```

Slika 4.17: Ansible: spreminjanje konfiguracijskih datotek

```
{% set configs = neutron.get('configs', [
{
  'path': '/etc/neutron/neutron.conf',
  'sections': {
    'database': {
      'connection': 'mysql://neutron:{0}@controller/neutron'.
          ↪ format(NEUTRON_DBPASS),
    },
    'DEFAULT': {
      'auth_strategy': 'keystone',
    }
  },
}, {
  'path': '/etc/neutron/plugins/ml2/ml2_conf.ini',
  'sections': {
    'ml2': {
      'type_drivers': 'gre'
      }
    }
}])%}
```

Slika 4.18: Salt: vrednosti za konfiguracijske datoteke

Salt Odločili smo se, da vrednosti ne bodo le nizi, ampak gnezdene zgoščene tabele (4.18), ki nazorno ponazarjajo sekcije v datotekah INI. To nam je omogočal sistem Jinja2 (4.19), ki ga uporablja Salt za datoteke z direktivami. Namreč znotraj Jinja2 podprtih datotek lahko uporabljaš programski jezik Python in zato je rokovanje z zgoščenimi tabeli precej olajšano. Uporaba pomožnega orodja je bila nesmiselna za Salt, saj omogoča pogojne klice znotraj ukazne lupine.

```
{% from "neutron/vars.sls" import configs %}

{% for config in configs %}
{% set path = config.path %}
{% for section, values in config.sections.iteritems() %}
{% for key, value in values.iteritems() %}

openstack-config {{path}} {{section}} {{key}}:
  cmd.run:
    - name: openstack-config --set {{path}} {{section}} {{key
      ↪ }} {{value}}
    - onlyif: '[[ $(openstack-config --get {{path}} {{section
      ↪ }} {{key}}) != {{value}} ]]'
    - require:
      - pkg: openstack-utils
      - pkg: openstack-keystone
    - require_in:
      - cmd: keystone_manage_sync
      - service: openstack-keystone
{% endfor %}
{% endfor %}
{% endfor %}
```

Slika 4.19: Salt: spreminjanje konfiguracijskih datotek

Opažanja Salt je zmogel opraviti vse delo, medtem ko smo za Ansible morali razviti pomožno orodje. Pri zapisu vrednosti smo pri Saltu dosegli boljšo preglednost od Ansibla.

4.2.7 Baza

Večina modulov, ki sestavljajo oblak, potrebuje za delovanje podatkovni strežnik. Na njem morajo, še preden začnejo obratovati, ustvariti podatkovno bazo, novega uporabnika in novemu uporabniku določiti administratorske pravice za novonastalo bazo. Izbrali smo strežnik MySQL ker je tudi izbran v navodilih, po katerih smo delali.

Primer Večina modulov ima isti postopek. Za primere smo izbrali modul *Nova*, lahko pa bi izbrali kateregakoli.

Ansible Zna delati z resursi oz. konstrukti, ki smo jih potrebovali - uporabniki, baze in privilegiji uporabnikov. Direktive, vidne na primeru 4.21, so zelo preproste. S prvo se zagotovi, da je baza ustvarjena. Z drugo se zagotovi, da je uporabnik ustvarjen in ima dodeljene pravice. Obe imata nekaj dodatnih atributov. Pri prvi smo dodali atributa za dostop do strežnika – uporabniško ime (`login_user`) in geslo (`login_password`). Če bi imeli veliko takšnih direktiv, bi nastalo nesmiselno podvajanje atributov za dostop, zato Ansible nudi alternativni način. Na ciljnem vozlišču je treba v domačem direktoriju ustvariti datoteko `.my.cnf` z vsebino, ki jo prikazuje primer 4.20, Ansible nato iz nje črpa informacije za dostop. To smo tudi storili pri drugi direktivi. Zadnja ima zanko `with_items` z dvema vsebinama, kar pomeni, da smo dejansko ustvarili dva uporabnika. Prvega za lokalne dostope in drugega za oddaljene dostope.

```
[client]
user=root
password={{ mysql_root_password }}
```

Slika 4.20: Ansible: alternativni način za dostop do strežnika

```
# Located: tasks/db.yml

- mysql_db:
    name=nova
    encoding=UTF8
    login_user=root
    login_password={{ mysql_root_password }}

- mysql_user:
    name=nova
    password={{ NOVA_DBPASS }}
    priv=nova.*:ALL,GRANT
    host={{item}}
  with_items:
    - localhost
    - '%'
```

Slika 4.21: Ansible: inicializacija podatkovne baze

Salt Tudi zna delati z istimi konstrukti kot Ansible, vendar je njegova rešitev bolj kompleksna (4.22). Berljivost je otežena, ker so direktive nekoliko razpihnjene, vsaka namreč zahteva zapis informacij za dostop do strežnika, alternativnega načina pa žal ni. Zaradi velikosti rešitve smo iz nje naredili makro in ga nato uporabili pri vseh modulih. Od Ansibla se razlikuje tudi v načinu dodeljevanja privilegijev. Ansible omogoča dodeljevanje v direktivi za uporabnike, medtem kot Salt zahteva ločeno direktivo.

Opažanja Uporaba makroja za Salt zelo olajša stvari, toda vseeno se ne more kosati z berljivostjo oz. razumljivostjo rešitve za Ansible.

```
# BEGIN: nova/db.sls
{% from "nova/vars.sls" import NOVA_DBPASS %}
{% from 'common/macros.sls' import openstack_setup_db %}

{{ openstack_setup_db('nova', NOVA_DBPASS) }}
# END: nova/db.sls

# BEGIN: common/macros.sls
{% import "common/vars.sls" as common with context %}

{% macro openstack_setup_db(project, password) %}
  {% for host in ['localhost', '\'%\''] %}
{{ project }}_user - {{ host }}:
  mysql_user.present:
    - name: {{ project }}
    - password: {{ password }}
    - host: {{ host }}
    - connection_host: localhost
    - connection_user: root
    - connection_pass: {{ common.mysql_root_password }}
    - require:
      - service: mariadb
  {% endfor %}

{{ project }}_database:
  mysql_database.present:
    - name: {{ project }}
    - connection_host: localhost
    - connection_user: root
    - connection_pass: {{ common.mysql_root_password }}
    - require:
      - mysql_user : {{ project }}

  {% for host in ['localhost', '\'%\''] %}
{{ project }}_privileges - {{ host }}:
  mysql_grants.present:
    - grant: all privileges
    - host: {{ host }}
    - database: {{ project }}.*
    - user: {{ project }}
    - connection_host: localhost
    - connection_user: root
    - connection_pass: {{ common.mysql_root_password }}
    - require:
      - mysql_database : {{ project }}
  {% endfor %}
{%- endmacro %}
# END: common/macros.sls
```

Slika 4.22: Salt: inicializacija podatkovne baze

4.2.8 Požarni zid

Spletni oblak OpenStack smo nameščali na operacijskem sistemu Fedora, ki za požarni zid uporablja servis *firewalld* namesto servisa *iptables*.

Primer Za modul Horizon je treba v požarnem zidu odpreti vrata 80 in 443 za spletni strežnik in vrata 6080 za dostop VNC do navideznih računalnikov.

Ansible Zna delati z resursem firewalld in zato ni bilo težav (4.23).

```
- firewalld:
    port={{item}}
    permanent=true
    state=enabled
  with_items:
    - 80/tcp
    - 443/tcp
    - 6080/tcp
```

Slika 4.23: Ansible: požarni zid

Salt Ne zna delati z resursom firewalld, toda zna delati z resursom iptables. Firewalld bi lahko trajno izključili in ga nadomestili z iptables, vendar tega nismo storili, saj bi se sicer oblaka razlikovala. Odločili smo se, da bomo požarni zid konfigurirali v ukazni lupini, kot je vidno na primeru 4.24.

Opažanja Oba omogočata konfiguriranje vsaj enega tipa požarnega zidu. V našem primeru smo uporabili prav tistega, ki ga Salt ne podpira, zato smo imeli z njim več dela.

4.3 Analiza orodij

Stroški za upravljanje konfiguracij se sestojijo iz stroškov za načrtovanje, postavljanjem naprav, uvajanjem sprememb in odpravljanjem napak. Načrtovanje se sestoji

```
{% for port in ['80/tcp', '443/tcp', '6080/tcp'] %}
firewalld open port - {{ port }}:
  cmd.run:
    - name: 'firewall-cmd --permanent --add-port={{ port }}'
    - unless: 'firewall-cmd --list-ports | egrep \s?{{ port
      ↪ }}\s?'
    - require:
      - service: firewalld
{% endfor %}
```

Slika 4.24: Salt: požarni zid

iz določanja želenega stanja in pisanja postopka, s katerim bomo dosegli stanje. Postavljanje naprav se sestoji iz kreiranja računalnikov z znano začetno konfiguracijo, ki jo lahko pozneje spreminjamo. Spremembe se uvede tako, da se spremeni trenutna konfiguracija naprav. Če so spremembe neustrezne, nas lahko pripeljejo do napak, ki jih je treba odpraviti.

Nekateri stroški so fiksni in jih težko nadzorujemo, medtem ko so drugi odvisni od odločitev sistemskega administratorja. Stroški načrtovanja so isti ne glede na način upravljanja sistema. Stroški postavitve so spremenljivi in odvisni od tega, ali smo naprave postavili ročno ali samodejno. Stroški odpravljanja težav so najbolj spremenljivi in zato predstavljajo priložnost za velike prihranke. So ključni dejavnik pri odločitvah administratorja [1].
Ena od največjih težav je nedeterministično uvajanje sprememb. Da bi zagotovili identično stanje na več računalnikih, je treba spremembe uvesti v istem vrstnem redu. Orodja CM to dosežejo z deterministčnim razvrščanjem direktiv in s tem zagotovijo, da so spremembe uvedene na vseh računalnikih enako. Ta vidik je zelo pomemben, saj je pri nepoznavanju točnega vrstnega reda težko napovedati obnašanje računalnikov. Posledično so zato manj zanesljivi. Zvišajo se tudi stroški dela, saj je treba več časa nameniti odpravljanju napak [2].

Orodje Salt omogoča štiri načine za razvrščanje direktiv, ki jih lahko sočasno uporabljamo [20]. Našteti so po prioriteti:

- Razvrščanje z navajanjem odvisnosti:
  Priporočen način, ki zahteva navajanje odvisnosti med direktivami. Če direktiva X potrebuje spremembe direktive Y in prav tako direktiva Z potrebujeje spremembe direktive X, potem bo Salt ugotovil, da je končni vrstni red: Z - Y - X.

- Eksplicitno razvrščanje:
  Direktivam lahko podamo položaj v končnem vrstnem redu. Če direktivi X določimo zadnji položaj, direktivi Y 1. položaj in direktivi Z 2. položaj, bo končni vrstni red: Y - Z - X.

- Pozicijsko razvrščanje:
  V tem primeru bodo direktive zasedle isti položaj kot v datotekah. Če je v datoteki najprej zapisana direktiva X, nato Z, na koncu pa še Y, bo končni vrstni red: X - Z - Y. Ta način je privzet le za začetno datoteko v vlogah, za preostale datoteke pa ga je treba vključiti v nastavitvah.

- Leksikografsko razvrščanje:
  Ne glede na to, kje imamo zapisane direktive, bodo direktive, ki niso zajete v predhodnih načinih, leksikografsko razvrščene. Torej bi imele direktive X, Y, Z končni vrstni red X - Y - Z. S tem so pri Saltu dosegli deterministično razvrščanje na vseh operacijskih sistemih.

Če ne bi imeli leksikografskega razvrščanja in če ne bi vključili pozicijskega ali eksplicitnega razvrščanja, bi nam preostalo le razvrščanje z navajanjem odvisnosti. Zadnje je edino nedeterministično in tu bi nastale težave. Zato so morali dodati leksikografsko razvrščanje, ki nam vedno zagotovi determinizem. Pri postavljanju spletnega oblaka smo v direktivah poskušali navesti vse potrebne odvisnosti, tiste, ki jih niso imele, so bile prepuščene leksikografskemu razvrščanju. Kadar smo pozabili na katero odvisnost, se je direktiva zaradi leksikografskega razvrščanja znašla na mestu, ki ga je določalo njeno ime. Včasih se je zaradi tega zalomilo, zato smo morali popraviti direktivo in nato ponoviti celoten postopek.

Orodje Ansible omogoča samo deterministično pozicijsko razvrščanje, ki ima druge težave. Zanj je treba poskrbeti, da so direktive napisane v pravilnem vrstnem

redu. Pri delu smo sledili navodilom. Za posamezni korak smo napisali direktivo in zagotovili, da si sledijo tako kot v navodilih. Zato nismo imeli nikakršnih težav z razvrščanjem.

Pri obeh orodjih smo kdaj pa kdaj naleteli na sintaksno ali semantično napako. Oba tipa napak se da hitro odpraviti z branjem dokumentacije, kar ne velja za napake zaradi razvrščanja. Te smo imeli le pri Saltu zaradi leksikografskega razvrščanja, njihova razrešitev pa nam je vzela kar nekaj časa.

Za preostale aspekte smo se odločili, da bi jih bilo najbolje prikazati v tabelah, zato smo jih razvrstili v tri tabele: v tabelo s prednostimi orodja Ansible (4.1), s prednostimi orodja Salt (4.2) in z njunimi skupnimi prednostmi (4.3).

Zdaj pa sledi razglasitev zmagovalca. Odločili smo se v prid orodja Ansible. Že iz števila aspektov ugotovimo, da je boljši, pomemben pa je predvsem eden – celotni čas postavitve. Pri Saltu smo veliko časa porabili zaradi njegovih pomankljivosti, kar je rezultiralo k enkrat daljšem času kot pri Ansiblu. Pri postavljanju je predvsem problematično leksikografsko razvrščanje in pomanjkanje mehanizmov za hitro odpravljanje napak, kar posledično vpliva na višino stroškov.

Aspekt	**Utemeljitev**
Preprostost sintakse	Z jezikom YAML poskušajo doseči vse in zato je tudi sintaksa preprostejša in preglednejša. Pri Saltu ni tako, saj uporabljajo kombinacijo dveh različnih jezikov - YAML in Jinja2.
Definiranje spremenljivk	Precendenčni vrsti red spremenljivk je izjemen, ko ugotoviš, kako deluje.
Uporaba vlog	Pri nameščanje orodja se namesti tudi upravljalec vlog (*role manager*). Z njim je možno na preprost način vključiti obstoječe vloge drugih uporabnikov. Salt ima tudi nekaj podobnega, vendar ni tako preprosto kot pri Ansiblu.
Dokumentacija	Resursi so bolje razloženi. Prav tako je tudi uradni priročnik bolje napisan od Saltovega.
Odpravljanje napak	Problematičnim direktivam lahko dodamo oznake (tags), npr. "ne_dela". Nato pri apliciranju določimo, da se upoštevajo le direktive s to oznako.
Nameščanje orodja	Namešča se ga samo na centralnem vozlišču.
Apliciranje direktiv	Brez težav. Pri Saltu smo včasih pri apliciranju izgubili stik s podvozlišči.
Celotni čas	Za Salt smo porabili enkrat več časa.

Tabela 4.1: Ansible: prednosti

Aspekt	**Utemeljitev**
Hitrost apliciranja	Za skalabilnost uporabljajo sporočilno vrsto ZeroMQ in posledično je zato hitrejši pri večjem številu podvozlišč. Ansible za vsako podvozlišče odpira povezavo ssh, kar se ne obnese ravno najbolje pri večjem številu.
Razširitev sintakse	Poudarek je na uporabi makrojev sistema Jinja2.
Pogojni klici	Pogoji se izvršijo v lupini.

Tabela 4.2: Salt: prednosti

Aspekt	**Utemeljitev**
Zbirka uradnih resursev	Oba znata delati z veliko znanimi resursi.
Pisanje novega resursa	Oba imata veliko primerov neuradnih resursev na spletnem repozitoriju GitHub, s katerimi si lahko pomagamo pri pisanju novega.
Pridobivanje dejstev (*facts*)	Oba znata preprosto pridobiti dejstva od podvozlišč.
Iteriranje	Pri Ansiblu se hitreje zapiše enostavnejše zanke. Salt ima drugo prednost. Omogoča zapisovanje bolj kompleksnejših zank. Za ta aspekt smo se odločili, da sta orodji izenačeni.

Tabela 4.3: Skupne prednosti

Poglavje 5

Sklepne ugotovitve

V diplomskem delu smo predstavili orodja za upravljanje konfiguracij in vzroke za njihov nastanek. Izbrali smo štiri orodja, ki so v zadnjem času medijsko bolj izpostavljena od drugih. Z vsakim smo postavili spletni strežnik in pri tem ocenili težavnost namestitve in pisanja direktiv. Nato smo izbrali dve orodji, ki sta bili boljši in s katerima nismo imeli težav. Z njima smo še postavili spletni oblak OpenStack. Pri tem smo ju podrobneje spoznali in zato smo lahko primerjali njune konstrukte in pristope, s katerimi smo se srečevali pri postavitvi. Glavni prispevek diplomske naloge je evalvacija orodij na praktičnih primerih. Z delanjem na primerih smo posledično dobili tudi delujočo rešitev za primer, ki smo si ga postavili - postavitev oblaka.

Z evalvacijo orodij smo olajšali izbiro vsem novim na področju upravljanja konfiguracij, ali tistim, ki želijo ročno upravljanje konfiguracij zamenjati z avtomatiziranim, ali tistim, ki imajo avtomatizirane rešitve, vendar z njimi niso zadovoljni. Z delom na oblaku OpenStack smo pokazali, kako avtomatizirati postavljenje velikega sistema z orodji CM. Na delujoči avtomatizirani rešitvi za postavljanje oblaka se lahko naučimo, iz česa je sestavljen in kako ga pravilno postaviti.

Diplomo bi lahko izboljšali z dodatnimi primeri, za katere bi napisali avtomatizirane rešitve. Tudi pri rešitvi za postavitev oblaka je še veliko možnosti za izboljšave. Lahko bi jo še razširili za postavitev na več računalnikih (multi-node).

Rešitev za Ansible bom gotovo uporabljal tudi sam, saj sem z njo zadovoljen.

V prihodnje bom vključil še preostale module oblaka (trove, heat, swift). O njeni uporabi bodo razmislili tudi v laboratoriju za računalniške komunikacije - LRK.

Literatura

[1] Burgess, Mark and Couch, Alva. Modeling Next Generation Configuration Management Tools. In Proceedings of the 20th Conference on Large Installation System Administration, 2006.

[2] Steve Traugott Why Order Matters: Turing Equivalence in Automated Systems Administration. In Proceedings of the 16th Conference on Large Installation System Administration, 2002.

[3] Thomas Delaet, Wouter Joosen, and Bart Van Brabant. A survey of system configuration tools. In Proceedings of the 24th Conference on Large Installation System Administration, 2010.

[4] Operacijski sistem Fedora. Dostopno na: http://fedoraproject.org/

[5] Scalable and Understandable Provisioning with Ansible and Vagrant. Dostopno na:
https://julien.ponge.org/blog/scalable-and-understandable-provisioning-with-ansible-and-vagrant/

[6] Orodje Puppet za upravljanje konfiguracij. Dostopno na:
http://puppetlabs.com/

[7] Orodje Chef za upravljanje konfiguracij. Dostopno na:
http://www.getchef.com/

[8] Orodje Salt za upravljanje konfiguracij. Dostopno na:
http://www.saltstack.com/

[9] Orodje Ansible za upravljanje konfiguracij. Dostopno na: http://www.ansible.com/

[10] Compute Engine Management with Puppet, Chef, Salt, and Ansible. Dostopno na: https://cloud.google.com/developers/articles/google-compute-engine-management-puppet-chef-salt-ansible

[11] Review: Puppet vs. Chef vs. Ansible vs. Salt. Dostopno na: http://www.infoworld.com/d/data-center/review-puppet-vs-chef-vs-ansible-vs-salt-231308

[12] Puppet vs. Chef vs. Ansible vs. Salt. Dostopno na: http://www.itworld.com/virtualization/383785/puppet-vs-chef-vs-ansible-vs-salt

[13] Puppet vs. Chef vs. Ansible vs. Salt. Dostopno na: http://www.networkworld.com/article/2172097/virtualizationuppet-vs–chef-vs–ansible-vs–salt/virtualization/puppet-vs–chef-vs–ansible-vs–salt.htm

[14] Orodje MCollective - The Marionette Collective. Dostopno na: http://puppetlabs.com/mcollective

[15] Orodje Vagrant. Dostopno na: https://www.vagrantup.com/

[16] Orodje VirtualBox. Dostopno na: https://www.virtualbox.org/

[17] Spletni oblak OpenStack. Dostopno na: http://www.openstack.org/software/

[18] OpenStack Installation Guide for Red Hat Enterprise Linux, CentOS, and Fedora. Dostopno na: http://docs.openstack.org/icehouse/install-guide/install/yum/content/

[19] Jezik Jinja2. Dostopno na: http://jinja.pocoo.org/docs/dev/

[20] One week of Salt: frustrations and reflections. Dostopno na: https://steveko.wordpress.com/2014/02/17/one-week-of-salt-frustrations-and-reflections/